Technology Consulting Company
Research, Development &
Global Standard

Interesting Issues During NVMe Driver Development

2018/11/28 @ BitVisor Summit 7

Ake Koomsin

Agenda

- NVMe Drive Degradation
- Copying Back Completion Entries Correctly
- Variety of NVMe controllers
- Troublesome Apple
- Strange Race Condition Between a Submission/Completion Doorbell Pair
- MSI-X Problem
- Troublesome UEFI Firmware

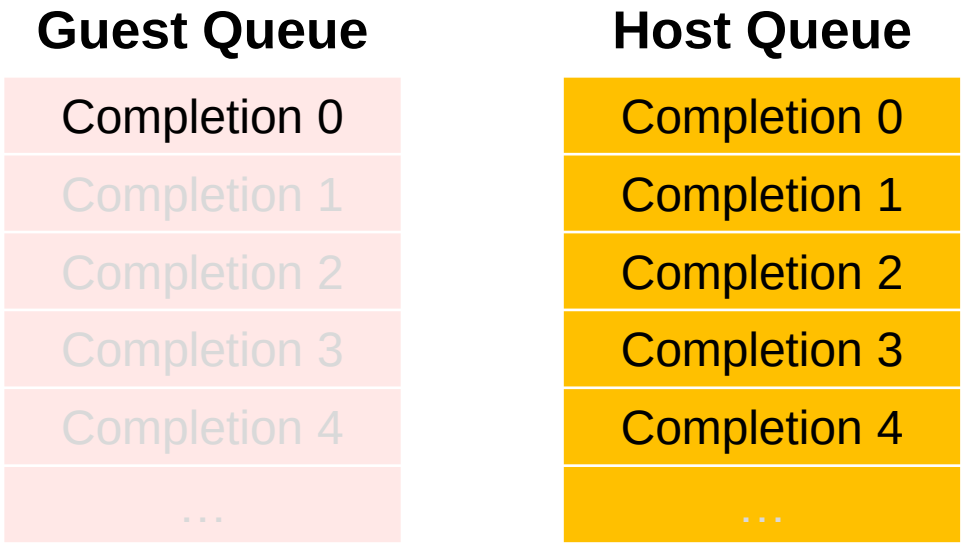
NVMe Drive Degradation

- This happened on one of our iMac that we (ab)use
- Its read/write performance dropped significantly
- Luckily, low-level formatting using `nvme-cli` helps
  - Format Command is a part of the standard (optional)
  - It is very fast

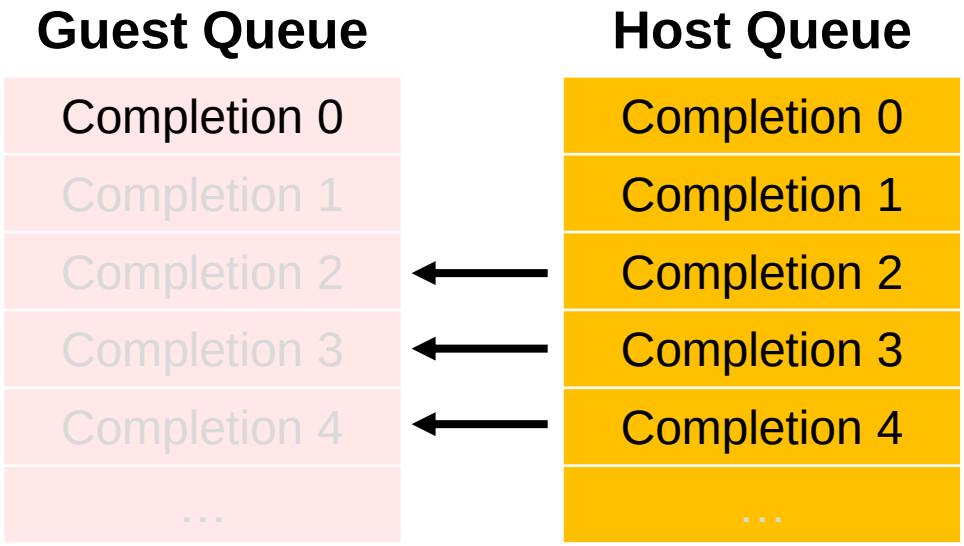
Copying Back Completion Entries Correctly

- This sounds simple, but a little bit tricky
- Copying back Completion Entries are done by CPU
  - beware of out-of-order execution

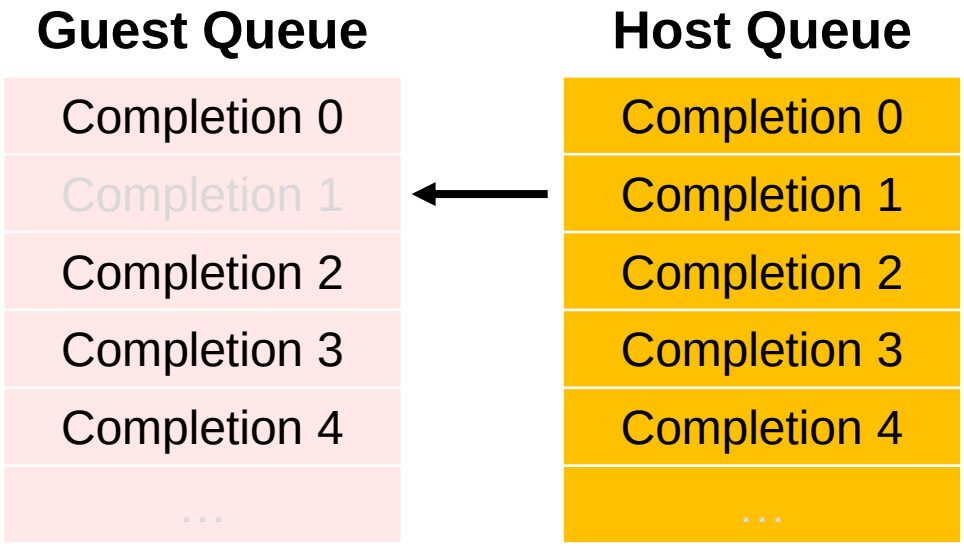
Copying Back Completion Entries Correctly



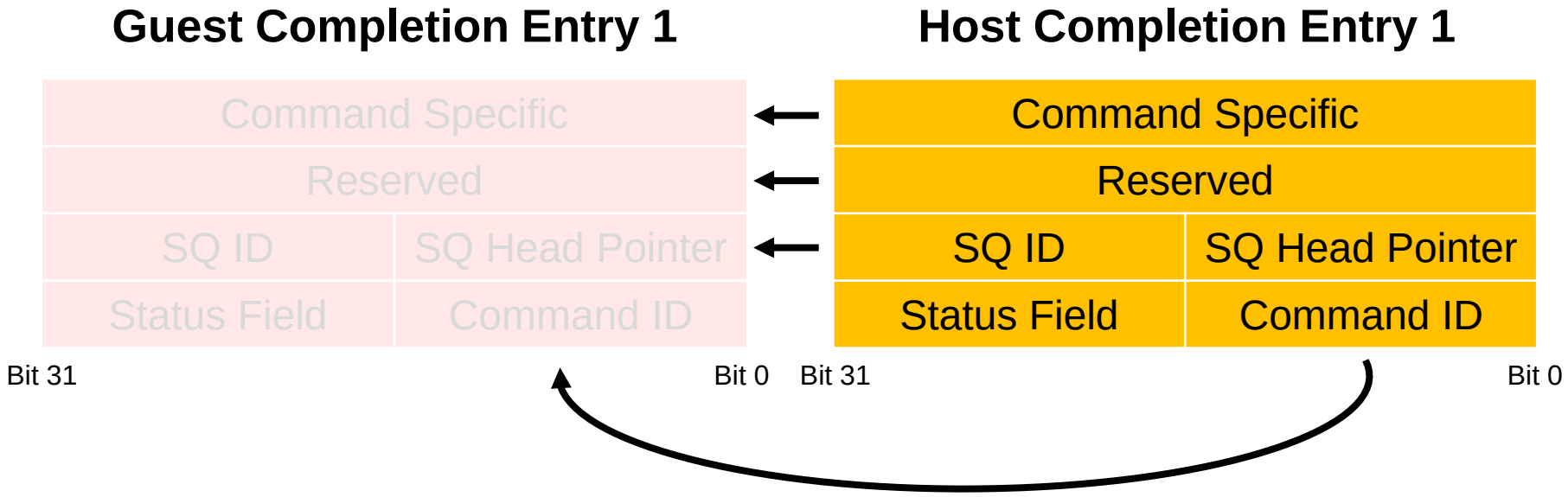
Copying Back Completion Entries Correctly



Copying Back Completion Entries Correctly



Copying Back Completion Entries Correctly



Copying Back Completion Entries Correctly

Guest Completion Entry 1

Command Specific	
Reserved	
SQ ID	SQ Head Pointer
Status Field	Command ID

Bit 31 Bit 0

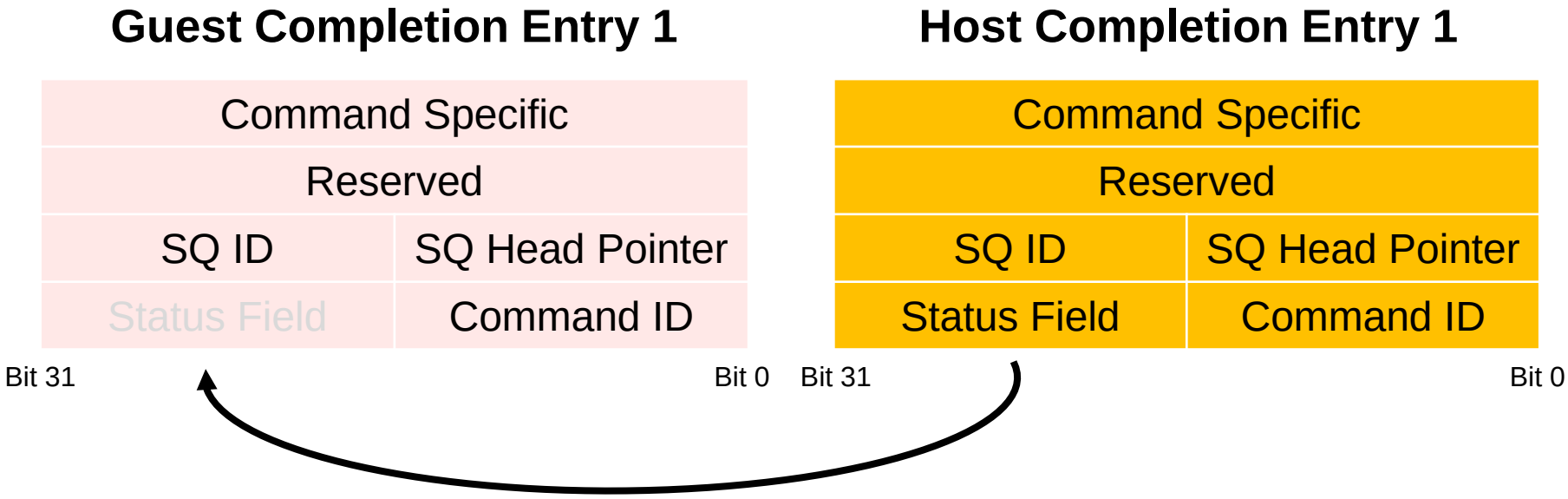
Host Completion Entry 1

Command Specific	
Reserved	
SQ ID	SQ Head Pointer
Status Field	Command ID

Bit 31 Bit 0

Before copying the
Status Field, sfence first

Copying Back Completion Entries Correctly



Copying Back Completion Entries Correctly

Guest Queue	Host Queue
Completion 0	Completion 0
Completion 1	Completion 1
Completion 2	Completion 2
Completion 3	Completion 3
Completion 4	Completion 4
...	...

Copying back is complete

Variety of NVMe controllers

- From one of notebook vendors we had our hands on, three types of NVMe controllers
  - Toshiba NVMe controller
  - Samsung NVMe controller
  - Intel NVMe Controller
- This kind of detail is not often listed in brochures…
- Apple uses either their own controllers or Samsung controllers

Variety of NVMe controllers

- Each controller is a little bit different
  - Number of queues supported
  - Maximum data transfer size
    - Toshiba: no limit
    - Samsung: 2 MiB
    - Some of Intel: < 512 KiB
    - Apple: 1 MiB
  - Supported optional commands
  - Quirks…
  - ETC

Troublesome Apple

- Our first Apple machine was MacBook
  - Apple customized NVMe controller
  - This Apple NVMe controller influences how NVMe driver is implemented
    - Due to its interesting “features”

Troublesome Apple

igel

NVMe Driver Development phases

Request shadowing

↓

Host and guest requests multiplexing

↓

Taking control of the controller
from UEFI firmware

Troublesome Apple

igel

- Host and guest requests multiplexing
  - The prototype driver did not work with Apple NVMe controller (as expected…)
  - The controller stalled, very painful to investigate
  - It turns out that Apple NVMe controller hates queuing up of Completion Entries
    - When the host receives an interrupt from the controller, the host should acknowledge Completion Entries from the controller immediately by writing to the corresponding Completion Doorbell
    - Commands from the hypervisor causes the problem because the prototype driver let the guest handles acknowledgement
  - In the end, have to multiplex in the time-sharing manner, either host-only requests or guest-only requests at a time

Troublesome Apple

igel

- Taking control of the controller from UEFI firmware
  - To do this, during NVMe driver initialization, we need to call UEFI DisconnectController() and let the firmware reconnect the controller
  - Sound simple, however it didn't work on this Macbook
  - Is something wrong with BitVisor implementation?
    - No, even DisconnectController()/ConnectController() from the UEFI shell does not work
  - Horay! There is a bug in the firmware

Troublesome Apple

- Taking control of the controller from UEFI firmware
  - Well, let's study the firmware's NVMe driver!
    - Found Apple firmware on Github
  - After studying the firmware NVMe driver for a little while, I noticed that the initialization code checks for various UEFI protocols
  - Is it possible that after DisconnectController(), some protocols remains causes ConnnectController() not to work properly?
  - The answer is yes. This leads to the commit "uefi: uninstall additional protocols after DisconnectController()"

Troublesome Apple

- Taking control of the controller from UEFI firmware
  - We were now able to reconnect the controller, however, it could not boot the OS. There were something more…
  - No choice but to dump all commands from the firmware
  - Apple-specific NVMe commands from the firmware!

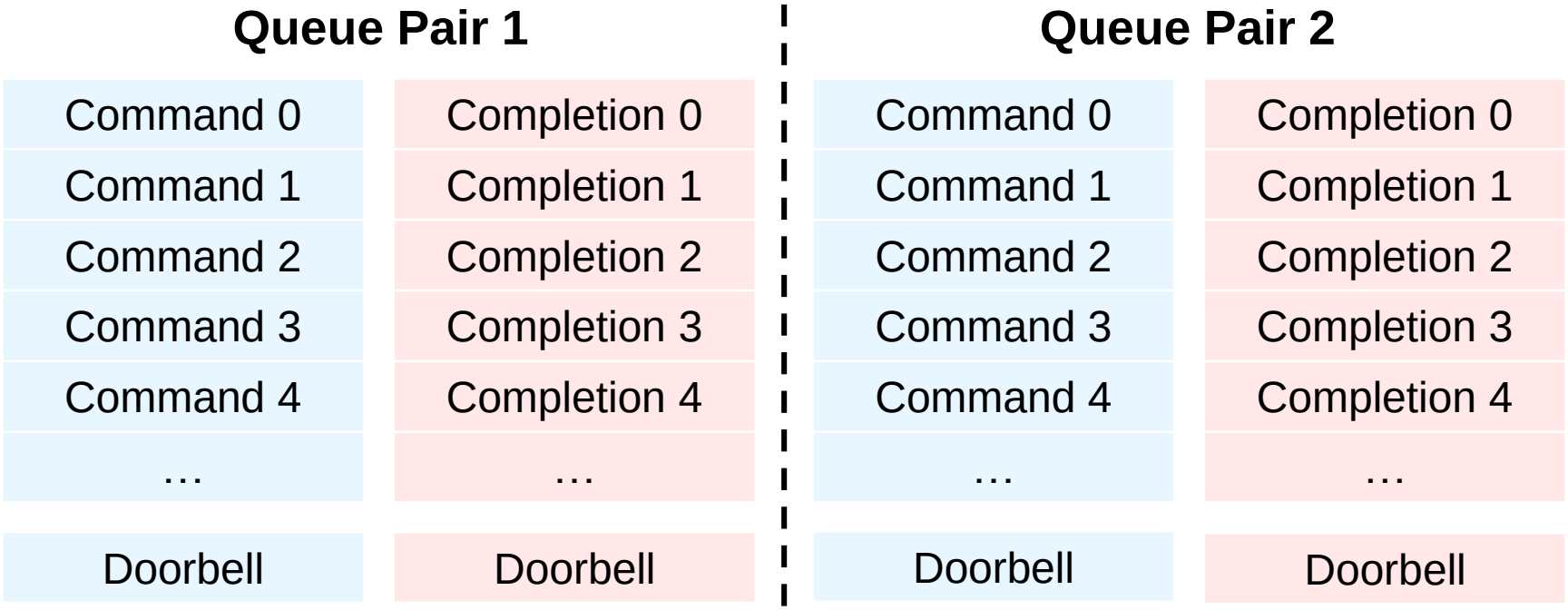
Troublesome Apple

- One more thing
  - According to our experiments, Apple NVMe controller hates reusing a data buffer
  - Let’s say you want to write zero to multiple areas. It is a good idea to create a zero buffer and keep reusing it, right?
  - This causes the controller to stall

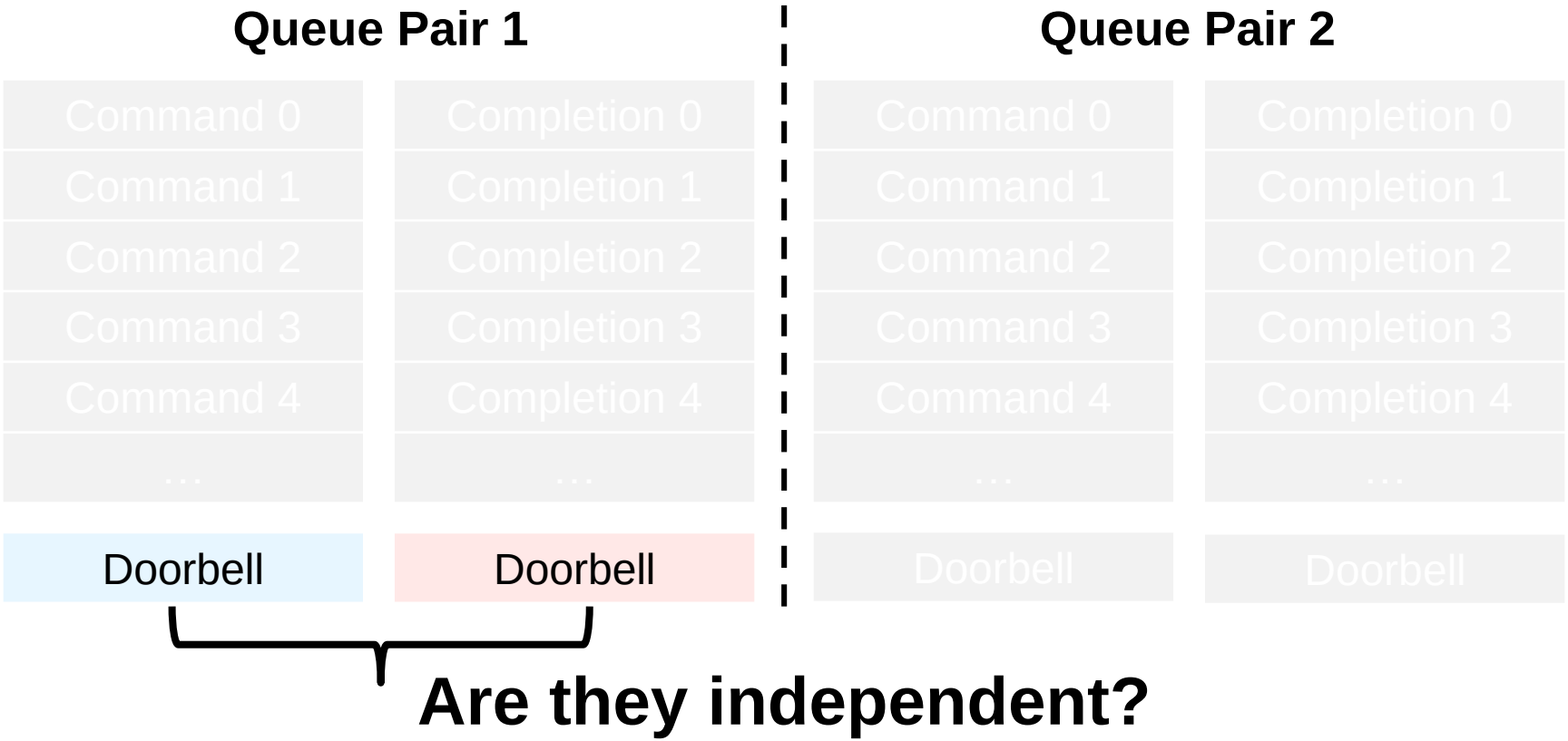
Strange race condition

- We had a problem with one of NVMe controller causing BSOD on Windows relating to watchdog timeout
- We made sure that only one thread can access either a submission queue part or a completion queue part at a time
- Each queue pair is independent from each other
- Data corruption was unlikely because there was no problem on Apple machines
- What could be the problem?

Strange race condition



Strange race condition



Strange race condition

- Strange race condition between a submission/completion Doorbell pair
  - For a given queue pair, the submission doorbell and the completion doorbell are not independent for this hardware
  - Adding a lock solves the problem

MSI-X Problem

- This only happens on one of Samsung OEM controllers
- It causes CentOS not to boot properly
- This does not happen to other controllers…

MSI-X Problem

- Concealing MSI-X solves the problem!
- Interrupt loss happens
- Initially, we thought that the current event sources are not enough
  - So, we added MSI-X mask clear as another event source
- Unfortunately, it was not enough
  - Can still reproduce the problem by boot and reboot repeatedly

MSI-X Problem

PCI: Flush MSI-X table writes

This patch fixes a kernel bug which is triggered when using the irqbalance daemon with MSI-X hardware.

Because both MSI-X interrupt messages and MSI-X table writes are posted, it's possible for them to cross while in-flight. This results in interrupts being received long after the kernel thinks they're disabled, and in interrupts being sent to stale vectors after rebalancing.

This patch performs a **read flush after writes** to the MSI-X table for mask and unmask operations. Since the SMP affinity is set while the interrupt is masked, and since it's unmasked immediately after, no additional flushes are required in the various affinity setting routines.

This patch has been validated with (unreleased) network hardware which uses MSI-X.

Revised with input from Eric Biederman.

Signed-off-by: Mitch Williams <mitch.a.williams@intel.com>
Acked-by: "Eric W. Biederman" <ebiederm@xmission.com>
Signed-off-by: Greg Kroah-Hartman <gregkh@suse.de>

MSI-X Problem

- In summary, read flush is required after clearing masks

Troublesome UEFI Firmware

igel

- Some of them don’t follow what the specification says
  - The spec says that the host software specifies number of queues it wants to use to the controller
    - The controller will then response with number of queues allocated
  - However, some firmware skip this
  - A little bit annoying for BitVisor as it doesn’t know how many queues it should allocate

Thank you